Arrays, Methods, and Memory

Arrays:

- An array is a way to store more than one value. It is a data structure that operates like a table.

Array Rules:

- Array size is fixed once designated.
- All data types within array must be the same.
- Use brackets [] when referring to values on an array.
- Arrays have scope just like variables.
- Data within the array is associated with an address.

```
public static void main(String[] args) {
    // TODO Auto-generated method stub


    int[] athlete = new int [5];
    int athleteLength = athlete.length;
    athlete[0] = 100;
    athlete[1] = 200;
    athlete[2] = 300;
    athlete[3] = 50;
    athlete[4] = 400;

    int athleteList = athlete[0];
    for (int athletes = 0; athletes < athleteLength; athletes++) {

    if (athlete[athletes] > athleteList);

}
}
```

An array, it contains 5 spaces each able to hold a value, all of them are integers.

Declaring & Initializing Arrays:

- Arrays are formed in a reference variable.
    - A reference variable is an address to a certain part of the array.
    - This reference variable as well as an array, is an object.

- <u>Declaration</u>

Datatype[] arrayName = new dataType [arraySize]

```
int[] athlete = new int [5];
```

- Datatype [] and new keyword
    - New – The keyword used in declaring an ***<u>object</u>***.

- Datatype [] arrayName – This will indicate the declaration of an array under a certain name – this is called a reference variable.
- Datatype [arraySize] – The data type within the array size must be an integer, memory will be allocated for the array size.

Initialization

- The initialization of an array resides within the array size. The integer value of the array size is allocated within the memory
- Memory can also be allocated dictated by a number list.

```
int[] athlete = {100, 200, 300, 400, 500};
```

This indicates an array in the form of a number list. These 5 values, (100,200,300,400,500) will be allocated in memory.

Arrays In Loops:

- When using arrays in loops, its best to use for loops.
  - For loops are loops that can be ran a set number of times.

```
for (int athletes = 0; athletes < athleteLength; athletes++) {

if (athlete[athletes] > athleteList);

System.out.println (athlete + " is the top athlete");
```

A for loop involving an array of athletes and finding the highest value.

- This loop will run if it has run less than the length of the array.
- This will run a check for every spot within the array comparing the values.
- Once it finds the highest value, it will print which spot in the array it is, and its value

Stack and Heap:

Memory Storage in Stack:

- Methods are stored within the stack
- Local variables are stored within the stack
- Addresses or reference variables are stored within the stack
- Primitive datatypes are stored on the stack

Heap:

- Objects are stored on the heap.
- Arrays are stored within the heap.
- Classes are stored within the heap.
- Non-primitive data types are stored on heap

```
public static void main(String[] args) {

    int[] athlete = {100,200,300,400,500};

    printContents(athlete);

    maxValue(athlete);

    minValue(athlete);

    total(athlete);
}
```

The array, athlete, is an object, and the data list assigned to it is allocated. In the main method, it is stored within the heap and utilizes methods.

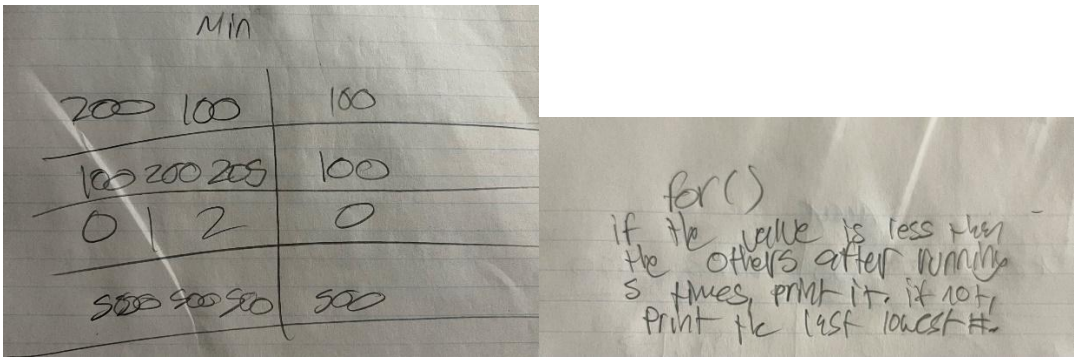
```
public static int maxValue (int[] athlete) {
    int maxV = athlete[0];
    int index = 0;
    for (int runs = 0; runs < athlete.length; runs++)
    if (athlete[runs] > maxV) {
        maxV = athlete[runs];
        index = runs; }
System.out.println("Max Value " + maxV + "Index: " + index);
return athlete[0];
```

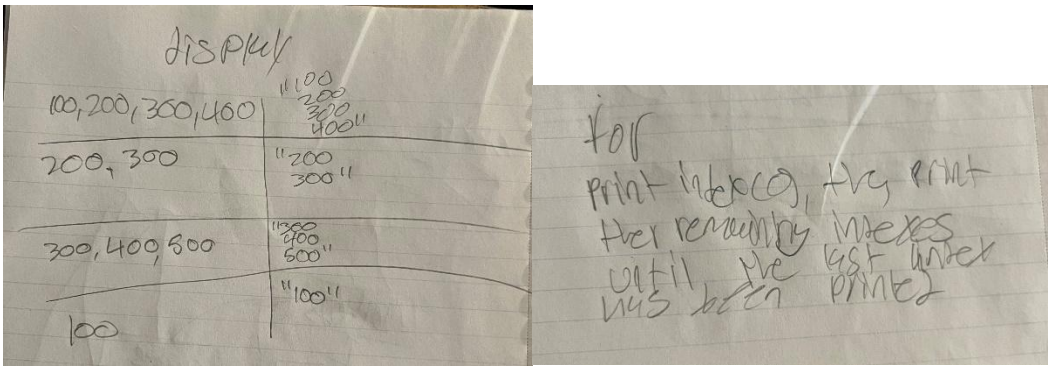
A method used to determine max value, this is stored on the stack, and after code is ran, it will dissipate.

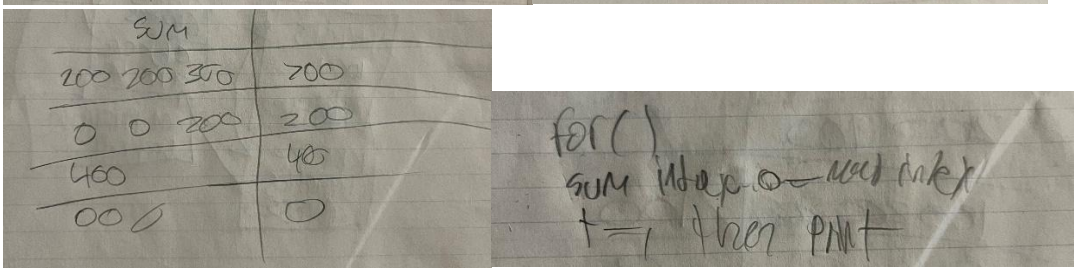
```
int maxV = athlete[0];
```

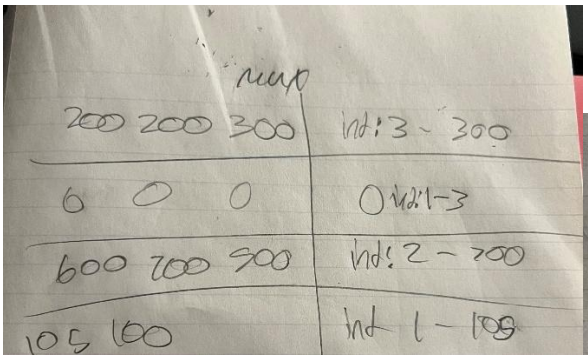
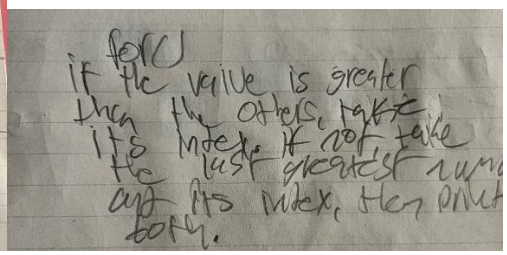
A reference variable. This variable references an address to a certain point on the array, this is stored on the stack.

Pseudo Code Examples: Top Athlete

1. 

2. 

3. 

4. 
max
200 200 300
ind: 3 - 300
6 0 0
0 ind: 1-3
600 200 500
ind: 2 - 200
105 100
ind 1 - 100

for()
if the value is greater
than the others, take
it's index. If not, take
the last greatest num
and its index, then print
both.

Pseudo Code Translation: Top Athlete

1.
```
public static int minValue (int[] athlete) {
    int minV = athlete[0];
    for (int runs = 0; runs < athlete.length; runs++)
    if (athlete[runs] < athlete[0])
        minV = athlete[runs];
        System.out.println("Min Value " + minV);
return athlete[0];
```

2.
```
public static int printContents (int[] athlete) {

    for (int runs = 0; runs < athlete.length; runs++)
    System.out.println(athlete[runs]);

return athlete[0];

}
```

3.
```
   public static int total (int[] athlete) {
   int sum = 0;
   for (int runs = 0; runs < athlete.length; runs++)
   sum += athlete[runs];
   System.out.println("Sum of values: " + sum);
   return athlete[0];
   }
}
```

4.
```
public static int maxValue (int[] athlete) {
    int maxV = athlete[0];
    int index = 0;
    for (int runs = 0; runs < athlete.length; runs++)
    if (athlete[runs] > maxV) {
        maxV = athlete[runs];
        index = runs; }
System.out.println("Max Value " + maxV + "Index: " + index);
return athlete[0];
```

2-D Arrays:

- Essentially an array inside of an array.
- Like a spreadsheet – rows and columns

<u>Declaration:</u>

- Int [] [] array = new int [size] [size]
    - [] [] – indicates 2-D array
    - [size 1] – rows [size 2] = columns

<u>Initialization:</u>

- Initialize with for loops
    - Works the same as a single array, but with a nested second for loop

```
        int[][] twoArray = new int [5][10];

    }

}


public static void arrayInitial (String[] twoArray[][]) {
    for (int set1 = 0; set1 < twoArray.length; set1++); {

    }
    for (int set2 = 0; set2 < twoArray.length; set2++);


}
```

Two nested for loops: deal with the first array [5] and second array [10]

Write and Read in Files:

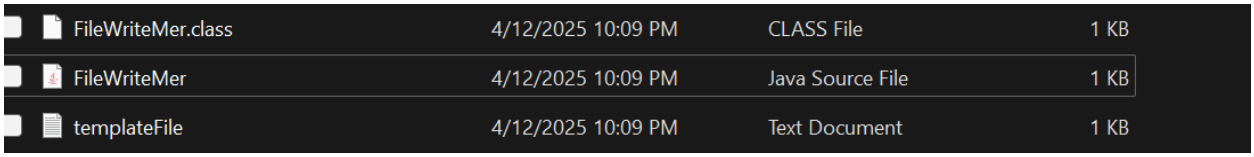
First, to read or write in files, the java.io package needs to be imported.

Reading and Writing:

- A scanner will be needed to read files, like a keyboard input.
    - Remember to close scanner
- A file must be in the same folder as the java file to be referenced.
- Files can be created; they are the datatype of object and use the “new” keyword.
- File names are strings

```
File example = new File ("examplefile");
Scanner scanFile = new Scanner(example);
scanFile.close();
```

Writing in a file



Files in same folder

```
import java.io.File;
import java.util.Scanner;
public class FileWriteMer {
```

Importing io and scanner package, both needed for the above file write

Passing Arrays, Primitive Data, Objects, and Methods:

Passing Primitive Data Types:

- Pass by value
    - Essentially, creating a copy of the value and passing it to a method.
    - Int numbers = 0 – datatype name = copy of value

Passing Objects:

- Pass by reference
    - Like pass by value, but instead, it uses an address as a value like an index
    - numbersArray[5] – array[index 5]

Stack Memory – Methods:

- Methods are stored on the stack
- They are called upon by parentheses – method(value) – updateRandomValue(number)\
- They are stored on the stack, and are allocated in memory as the code runs
- Once the code stops, the method then disappears

```java
public static void updateRandomValue (int passedArray[], int passedNumber) {
        Random randomizer = new Random();
        passedNumber = randomizer.nextInt(10) * 1;
        for (int runs = 0; runs > passedArray.length; runs++) {
            passedArray[runs] = randomizer.nextInt(10) * 1;
        }
}
}
```

```java
 */
import java.util.Random;
public class M04PassArrays {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int[] numbersArray = new int [10];
        int number = 0;
        System.out.println(updateRandomValue(numbersArray));
        System.out.println(updateRandomValue(number));
    }
```